DETAILED ACTION

1. Applicant's amendment dated February 25, 2008, responding to the Office action mailed September 25, 2007 provided in the rejection of claims 1-9, 11-16 and 18-24, wherein claims 1, 7, 13-14 and 19 are amended.

Claims 1-9, 11-16 and 18-24 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims currently amended have been fully considered but are moot in view of the new grounds of rejection – see *Applicant's Admitted Prior Art (AAPA)* - art made of record, as applied hereto.

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

2. Claims 1-5, 8-9, 11, 14-16, 19, and 21-24 are rejected under 35 U.S.C. 103(a) as being unpatentable over Cytron et al., (*Efficiently Computing Static Single Assignment Form and the Control Dependence Graph, 1991, ACM*) (hereinafter 'Cytron') in view of Applicant's Admitted Prior Art (hereinafter 'AAPA' - art made of record)

3. **As to claim 1** (Currently Amended), Cytron discloses (currently amended) a method for static single assignment form dead code elimination, the method comprising:

- examining a first instruction of a machine code off of a worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered) in memory;
- examining at least one second instruction, wherein the at least one second instruction is a source of the first instruction and; determining if any components within a particular field of the at least one second instruction are required (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch);
- when no components of the at least one second instruction are required, deleting the at least one second instruction from the machine code (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches); and
- when any component of the at least one second instruction is required, adding the at least one second instruction to the worklist in the memory (e.g., P. 480, 1st

Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered)

Cytron discloses Static Single Assignment Form Algorithm (e.g., Sec. 3 – Static Single Assignment Form) but does not explicitly disclose:

- wherein each of the first instruction includes a previous link and a write mask; and
- wherein each of the at least one second instruction includes a previous link and a write mask

However, AAPA discloses:

- wherein each of the first instruction includes a previous link and a write mask; and
- wherein each of the at least one second instruction includes a previous link and a write mask (e.g., [004] …by adding a single bit to each instruction, wherein that bit is designated as a live bit ... walking backwards over SSA link from use to definition …; [0007] - using this bit mask … during the backward walk over the SSA links …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of AAPA into the Cytron's system to further provide the following in the Cytron system:

- wherein each of the first instruction includes a previous link and a write mask; and

- wherein each of the at least one second instruction includes a previous link and a write mask

The motivation is that it would further enhance the Cytron's system by taking, advancing and/or incorporating the AAPA's system which offers significant advantages for an efficient scheme, since adding an instruction to the worklist can occur at most only one time and also designates all dead code as once suggested by AAPA (e.g., [0005])

4. **As to claim 2** (original) (incorporating the rejection in claim 1), Cytron discloses the method further comprising: generating the worklist by:

- for each of a plurality of instructions, determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- if the instruction is a critical instruction, writing the instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

5. **As to claim 3** (original) (incorporating the rejection in claim 2), Cytron discloses the method further comprising: setting a live bit for each component of the plurality of instructions (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

6. **As to claim 4** (Previously Presented) (incorporating the rejection in claim 2), Cytron discloses the method wherein each critical instruction is an instruction that generates an export value (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

7. **As to claim 5** (Previously Presented) (incorporating the rejection in claim 2), Cytron discloses the method further comprising: prior to generating the worklist:

receiving a plurality of instructions; adding to each instruction the previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions – applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences); and adding to each instruction the write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live).

8. **As to claim 8** (Previously Presented), Cytron discloses a method for static single assignment form dead code elimination comprising:

- receiving a plurality of instructions; (e.g., Sec. 2 Control Flow Graphs, 1st Par. - the statements of a program are organized into basic blocks ...);

generating a worklist in memory by:

- for each of the plurality of instructions, determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional

branch, and there are statements already marked live that are control dependent on this conditional branch); and

- if the instruction is a critical instruction, writing the instructions to the worklist in the memory (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).
- receiving a plurality of instructions; adding to each instruction a previous link (e.g., Sec. 6 – Construction of Control Dependences, 1st Par. – we show that control dependences are essentially the dominance frontiers in the reverse graph of the control flow graph; P. 477, 2nd Par. – the reverse control flow graph RCFG has the same nodes as the control flow graph CFG, but has an edge Y → X for each edge X → Y in CFG; the roles of Entry and Exit are also reverse; Sec. 9.3 – Conclusions – applying the early steps in the SSA translation to the reverse graph is an efficient way to compute control dependences);

adding to each instruction a write mask (e.g., P. 480, 1st Par., 1st point – Live(S) indicates that state S is live);

Cytron discloses Static Single Assignment Form Algorithm (e.g., Sec. 3 – Static Single Assignment Form) but does not explicitly disclose:

- adding to each instruction a previous link;
- adding to each instruction a write mask

However, AAPA discloses:

- adding to each instruction a previous link;

- adding to each instruction a write mask (e.g., [004] …by adding a single bit to each instruction, wherein that bit is designated as a live bit ... walking backwards over SSA link from use to definition …; [0007] - using this bit mask … during the backward walk over the SSA links …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of AAPA into the Cytron's system to further provide the following in the Cytron system:

- adding to each instruction a previous link;
- adding to each instruction a write mask

The motivation is that it would further enhance the Cytron's system by taking, advancing and/or incorporating the AAPA's system which offers significant advantages for an efficient scheme, since adding an instruction to the worklist can occur at most only one time and also designates all dead code as once suggested by AAPA (e.g., [0005])

9. **As to claim 9** (Previously Presented) (incorporating the rejection in claim 8), Cytron discloses the method of claim 8 further comprising: setting a live bit for each component of the plurality of instructions:

- examining a first instruction off of the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered);
- examining at least one second instruction in the machine code, wherein the at least one instruction is a source of the first instruction (e.g., P. 479, 2nd Par., 1st

point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch);

- determining if any component within a particular field of the at least one second instruction is live (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- when no components of the at least one second instruction are live, deleting the second instruction from the worklist (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches).

10. **As to claim 11** (Previously Presented) (incorporating the rejection in claim 8), Cytron discloses the method wherein each critical instruction is an instruction that

generates an export value (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

11. **As to claim 14** (Currently Amended), Cytron discloses an apparatus for static single assignment form dead code elimination comprising: at least one memory device storing a plurality of executable instructions of a machine code; and at least one processor operably coupled to the at least one memory device, operative to receive the plurality of executable instructions such that the at least one processor, in response to plurality of executable instructions is further operative to:

- examine a first instruction off of a worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered);
- examine at least one second instruction of the machine code; determine if any component within a particular field of the at least one second instruction is live (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements

already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and

- when no components are live, delete the second instruction from the machine code (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches)

Cytron discloses Static Single Assignment Form Algorithm (e.g., Sec. 3 – Static Single Assignment Form) but does not explicitly disclose:

- wherein the first instruction includes previous link and a write mask; and
- wherein the at least one second instruction is a source of first instruction and each of the at least one second instruction includes a previous link and a write mask

However, AAPA discloses:

- wherein the first instruction includes previous link and a write mask; and
- wherein the at least one second instruction is a source of first instruction and each of the at least one second instruction includes a previous link and a write mask
- (e.g., [004] …by adding a single bit to each instruction, wherein that bit is designated as a live bit ... walking backwards over SSA link from use to definition …; [0007] - using this bit mask … during the backward walk over the SSA links …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of AAPA into the Cytron's system to further provide the following in the Cytron system:

- wherein the first instruction includes previous link and a write mask; and
- wherein the at least one second instruction is a source of first instruction and each of the at least one second instruction includes a previous link and a write mask

The motivation is that it would further enhance the Cytron's system by taking, advancing and/or incorporating the AAPA's system which offers significant advantages for an efficient scheme, since adding an instruction to the worklist can occur at most only one time and also designates all dead code as once suggested by AAPA (e.g., [0005])

12. **As to claim 15** (Previously Presented) (incorporating the rejection in claim 14), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: generate the worklist by:

- for each of a plurality of instructions, determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements

already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and

- if the instruction is a critical instruction, writing the instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

13. **As to claim 16** (Previously Presented) (incorporating the rejection in claim 15), Cytron discloses the apparatus wherein critical instruction is an instruction that generates an export value (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch)

14. **As to claim 19** (Currently Amended), Cytron discloses an apparatus for static single assignment form dead code eliminations comprising: at least one memory device storing a plurality of executable instructions of a machine code; and at least one processor operably coupled to the at least one memory device, operative to receive the plurality of executable instructions such that the at least one processor, in response to the executable instructions is further operative to:

- receive a plurality of instructions; (e.g., Sec. 2 Control Flow Graphs, 1st Par. - the statements of a program are organized into basic blocks ...); and
- generate a worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered) by:
- for each of the plurality of instructions; determining if the instruction is a critical instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch); and
- if the instruction is a critical instruction, writing the instructions to the worklist:
- examine a first instruction off of the worklist;
- examine at least one second instruction from the machine code, wherein the at least one second instruction is a source of the first instruction (e.g., P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch);

- determine if any component within a particular field of the at least one second instruction is live; and when no component is live, delete the second instruction from the machine code (e.g., Fig. 17 – Dead code elimination; P. 479, last Par. – our algorithm, given in Figure 17, goes one step further in eliminating dead conditional branches).

Cytron discloses Static Single Assignment Form Algorithm (e.g., Sec. 3 – Static Single Assignment Form) but does not explicitly disclose to:

- add to each instruction a previous link; and
- add to each instruction a write mask

However, AAPA discloses to:

- add to each instruction a previous link; and
- add to each instruction a write mask
- (e.g., [004] …by adding a single bit to each instruction, wherein that bit is designated as a live bit ... walking backwards over SSA link from use to definition …; [0007] - using this bit mask … during the backward walk over the SSA links …)

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of AAPA into the Cytron's system to further provide the following in the Cytron system:

- add to each instruction a previous link; and
- add to each instruction a write mask

The motivation is that it would further enhance the Cytron's system by taking, advancing and/or incorporating the AAPA's system which offers significant advantages for an efficient scheme, since adding an instruction to the worklist can occur at most only one time and also designates all dead code as once suggested by AAPA (e.g., [0005])

15. **As to claim 21** (Previously Presented) (incorporating the rejection in claim 15), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of executable instructions is further operative to set a live bit for each component of the plurality of instructions (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch).

16. **As to claim 22** (Previously Presented) (incorporating the rejection in claim 9), Cytron discloses the method further comprising: when any component of the at least one second instruction is live (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference

parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch), adding the at least one second instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

17. **As to claim 23** (Previously Presented) (incorporating the rejection in claim 14), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: when any component of the at least one second instruction is live (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch), add the at least one second instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered)t.

18. **As to claim 24** (Previously Presented) (incorporating the rejection in claim 19), Cytron discloses the apparatus wherein the at least one processor in response to the plurality of instructions executable instructions is further operative to: when any

component of the at least one second instruction is live (e.g., Sec. 7.1 – Dead Code Elimination, 2nd Par.; P. 479, 2nd Par., 1st point through 3rd point – the statement is one that should be assumed to affect program output, such as an I/O statement, an assignment to a reference parameter, or a call to a routine that may have side effects; the statement is an assignment statement, and there are statements already marked live that use some of its outputs; the statement is a conditional branch, and there are statements already marked live that are control dependent on this conditional branch)., add the at least one second instruction to the worklist (e.g., P. 480, 1st Par., 3rd point – *WorkList* is a list of statements whose live-ness has been recently discovered).

Allowable Subject Matter

19. Claims 6-7, 12-13, 18, and 20 are objected to as being dependent upon a rejected base claim, but would be allowable if rewritten to overcome all the limitations of the base claim and any intervening claims.

The following is an examiner's statement of reasons for allowance:

20. **Regarding claims** 6-7, 12-13, 18, and 20, prior art of record fails to reasonably show or suggest "the write mask is a multi-bit field representing a number of components in a superword register; each of the plurality of instructions are written to the worklist a predetermined number of times, wherein the predetermined number of times is based on the number of components in the superword register".

Conclusion

21. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/Ben C Wang/
Examiner, Art Unit 2192
June 4, 2008

/Tuan Q. Dam/
Supervisory Patent Examiner, Art Unit 2192